QUICKBASIC COMMAND LIBRARIES

Enhance your QuickBASIC *programming environment with a library of your own favourite routines. Using examples from the* SuperDisk, *Ron Smith shows you just how simple it really is.*

All modern compilers make use of libraries and *QuickBASIC* is no exception. There are too many commands in *QuickBASIC* to include the code for them all in the program itself, it would run out of memory, so they're farmed out to libraries, of which there are three types: the main libraries, which are always searched when a program is compiled, optional libraries and user libraries. *QuickBASIC*'s main libraries (BCOM45.LIB and BRUN45.LIB) contain the code for the common commands like PRINT and VAL, but it makes sense for routines that are used rarely to be kept in a separate library.

Microsoft provides one optional library with *QuickBASIC*, QB.QLB. This contains code needed to call MS-DOS interrupts directly. It has been separated from the main libraries because few people use interrupts directly; *QuickBASIC* only searches the optional and user libraries if you tell it.

Unlike optional libraries, a user library is created by the user. It will usually contain a collection of specialised functions – like a statistics library – or it could just be a collection of those routines you find yourself using repeatedly. Whatever its purpose your user library is created in the same way.

Writing a library is very similar to writing any other program in *QuickBASIC*. Normally you have a main program and, perhaps, some SUBs or FUNCTIONS. But a user library has no main program code, only SUBs or FUNCTIONs which are called from a separate program in the same way as more familiar commands, like PRINT.

SUBs and FUNCTIONs are very similar. They both take parameters (a variable passed to the routine from the calling program) and assign them to temporary local variables (see *Ephemerals*, page 234). Both kinds of sub-routine can alter local variables but only FUNCTIONs will return a new version of a passed value to the calling program. As well as returning values FUNCTIONs are faster than SUBs.

Start *QuickBASIC* as if you were about to write a new program. Instead of beginning to write a main program go to the Edit menu and choose 'New FUNCTION'. *QuickBASIC* will prompt for a name and place you at the beginning of the new sub-routine. The only thing on screen will be the code marking the start and the end:

```
FUNCTION dummyfunc
END FUNCTION
```

At this stage, the sub-routine does nothing since it contains no code and there are no variables for it to act on. Let's consider these variables. A FUNCTION can be passed any number of variables of almost any type (but see below). You must specify all of them after the name of the sub-routine in a Variable List – a list of the variables passed to the sub-routine from the calling program. These are surrounded by brackets and will look like this:

```
FUNCTION dummyfunc(a,b,c,d,answers())
```

A variable list contains variables defining numbers, strings and arrays. Due to a limitation in *QuickBASIC*, you cannot pass a user defined type (TYPE...END TYPE) to a sub-routine in a library. Returning more than one value from a sub-routine requires an array.

The names of the variables in your main calling program need not be the same as the names in the variable list. Remember, they are usually local to the sub-routine – they're not the same as the variables in your main program, they merely have the same value. However, you must make sure that the variables are passed from the calling program in the same order that your routine is expecting them. If they are not then the variables in the sub-routine will get mixed up and it will act unpredictably.

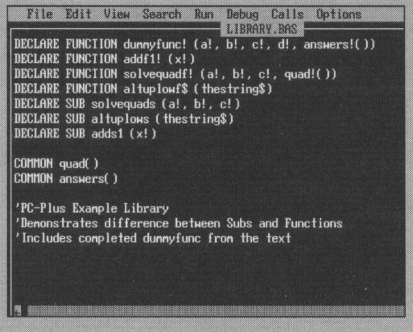
Our example routine 'dummyfunc' takes four integer variables and the name of an array to pass back the results. It adds the first and the last pair of numbers together, places the answers in the array and returns it to the calling program. We must tell it to expect the five variables in the variable list above. *QuickBASIC* knows answers() is an array because of the brackets after it.

Before we can use the array answers(), we must declare it as COMMON in the main module of the library. Being COMMON doesn't mean it spits on the pavement, but that it can be accessed from any point in either the calling program or the library. Make answers() COMMON like so:

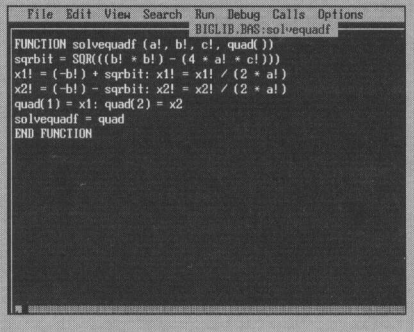
```
COMMON answers()
```

It is unnecessary to put the size of the array in the brackets. Having sorted out

COMPILING QUICKBASIC



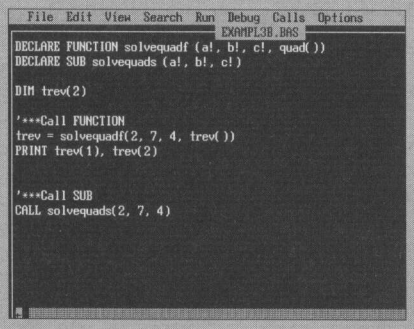
● The Main Module of the user library contains just the DECLARE statements for the sub-routines and the COMMON statements for shared variables



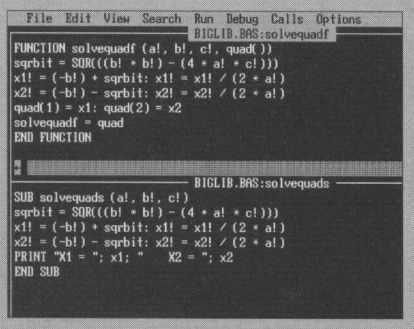
● FUNCTIONS in the user library are edited individually. *QuickBASIC* shows each SUB or FUNCTION as a separate entity

PROGRAMMING

SUBS AND FUNCTIONS IN QUICKBASIC



● SUBs must always be called from their own line, as shown, whereas FUNCTIONs are always part of an expression, as illustrated in this example code



● Here are a SUB and FUNCTION that perform basically the same job. The difference is that a SUB can't communicate its results to the calling program

PROGRAMMING

the variables we must next write the code in the sub-routine. In this case it just has to add the first and second pairs of numbers and place the two totals in the array, as follows:

```
tempa = a + b
tempb = c + d
answers(1) = tempa
answers(2) = tempb
```

Once the FUNCTION has performed the calculations on the values passed to it, we want it to return the answers to the calling program. Oddly, we do this by assigning the variable containing the answers (the array in this case) to the name of the FUNCTION:

```
dummyfunc = answers
```

The array answers() will now be returned to the calling program. If we fail to include this line then the array will not be returned. Our completed library function looks like this: (the first two lines of this must be typed as a single line).

```
FUNCTION dummyfunc(a, b, c, d,
answers())
tempa = a + b
tempb = c + d
answers(1) = tempa
answers(2) = tempb
dummyfunc = answers
END FUNCTION
```

We're nearly there now. However, before the library with our new FUNCTION is compiled, the FUNCTION must be declared. What this means is simply that in the main module of the library (and at the start of any program that will use the library), a DECLARE command must be present telling *QuickBASIC* about the library routines. Furthermore, this command must include the name of the sub-routine along with its variable list. If there is more than one sub-routine in the library each must have a DECLARE statement, like so: (these two lines must be typed as a single line).

```
DECLARE FUNCTION dummyfunc(a, b, c,
d,answers())
```

To save and compile the library go to the 'Run' menu and choose the 'Make Library' option. You will be prompted to supply a library name. Keep this the same as the name of the saved source code so, if you decide to edit the library, you know which source code to look at.

In order to try out your new library you must first return to MS-DOS. If you look at the *QuickBASIC* directory listing, you will find two libraries with the same name. One has the extension .QLB and the other has .LIB. The .QLB library is the one *QuickBASIC* uses while you develop your programs interactively. The other one is used when *QuickBASIC* compiles your program.

As mentioned, *QuickBASIC* only uses optional and user libraries when told to, this includes your new one. To have your new functions available you must now re-start *QuickBASIC* using the /L switch. This switch tells *QuickBASIC* to load a library:

```
QB /L libraryname
```

Your program will now have access to all the sub-routines in the loaded library. But remember, before you are able to use them you must DECLARE them at the top of your program in the same way that we did inside the library.

Right, you've loaded the new user library with the /L switch, you've DECLAREd the library function and you're now writing a program. To use a library FUNCTION you must assign the FUNCTION name and the variables to be passed (in the same order that your routine will expect them) to the variable in which you want the answers to be returned. This variable will be either a number, a string or an array.

If you're using an array to pass or return values you must first define the array in the calling program. The name of the array you create does not have to match the name in the DECLARE statement but it must be the name of the array that you will assign the answer to. For example, to use the FUNCTION dummyfunc in a real program you might do: (the first two lines must be typed as a single line).

```
DECLARE FUNCTION dummyfunc(a, b, c, d,
answers())
DIM results(2)
results = dummyfunc(1, 2, 3, 4, results())
```

If you are passing numerical values to a library sub-routine you must ensure that they are of the same type in the calling program and the library FUNCTION. If the sub-routine works solely with integers and your calling program works solely with single length floating point numbers, the variables must be converted to integers before being passed. You can force conversion in the calling command like so:

```
results = demofunction(a%, b%, c%)
```

Where a, b and c will all be converted to integers before demofunction is called. Now it's time to have a go yourself. To get you started, an example library with dummyfunc and several other example FUNCTIONS (with equivalent SUBS) has been included on the *SuperDisk* in the MAGAZINE directory. ●

EPHEMERALS

QuickBASIC **will pass variables by reference or by value. By default *QuickBASIC* passes variables by reference. When you pass a variable by value, only a copy of the variable is passed and all changes are made to the copy. This copy is lost when the routine exits. Passing by value makes for less problems in a complex program. Variables are passed by value when they are enclosed in brackets. All the examples in this article and in the example library on the *Superdisk* pass variables by value.**

The only problem with this is that, under some circumstances, it slows a program down. *QuickBASIC* has to make a temporary variable in another memory location and copy the original variable into it. It then passes the address of the temporary variable to your routine. If you are passing a 10K array then this process takes an appreciable time.

When you pass a variable by reference you pass the address of the original variable. Any changes that the sub-routine makes to the variable are permanent. In fact the variable in the original program will never be the same again.